

# Google Cloud のデータ パイプライン サービスの紹 介と選び方

Google Cloud Japan  
Data Analytics Specialist  
西村 哲徳



# アジェンダ

- Google Cloud の Smart Analytics Solutions
- データ パイプライン サービスを選択する
- Google Cloud のデータパイプライン サービス

**Google Cloud  Smart  
Analytics Solutions**

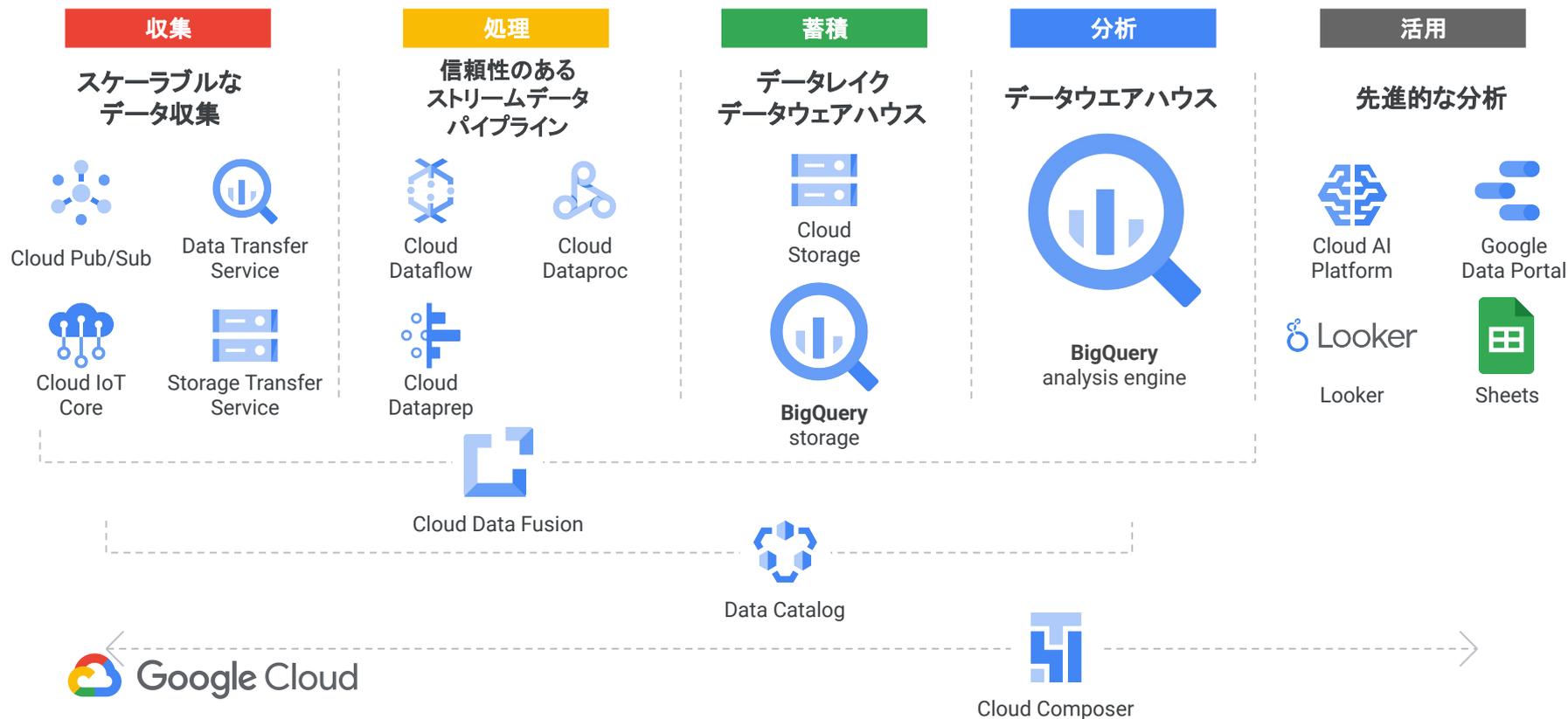


# Smart Analytics によるサイロ化したデータの統合

リアルタイムに、シームレスに、未来を予測。



# エンドツーエンドのデータ分析プラットフォーム



データパイプライン  
サービスを選択する



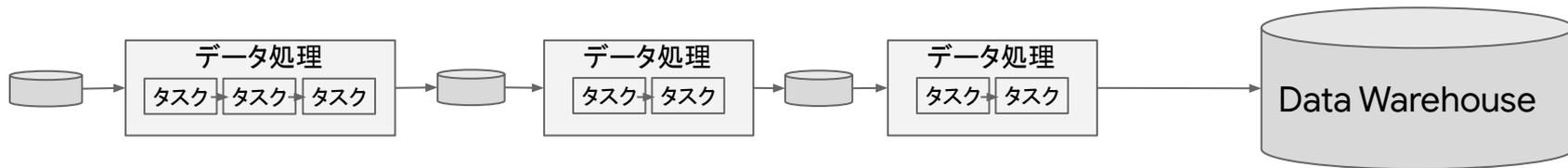
# データパイプラインとは？

- データソースからデータを収集して、クレンジング、エンリッチメント、各種チェック等の変換処理をして分析可能な形にして格納する一連のデータ処理のこと
- データパイプラインを構成する要素
  - データソース / シンク
    - ソース：業務アプリやウェブ、モバイル等のデータ
    - シンク：分析用に処理されたデータ、中間データ
    - 主に RDBMS、NoSQL、メッセージング、ストレージ サービス、SaaS 等
  - データ処理
    - データの変換処理を実行(ジョブ、タスク)
  - ワークフロー
    - 依存関係のあるデータ処理の実行順序を制御、管理
    - スケジュール
- 製品の観点では、データ処理製品にワークフローの機能が含まれていることが多い

# データ処理の方式 ETL と ELT

E(抽出)、T(変換)、L(ロード)

ETL: 抽出したデータを変換してからデータ ウェアハウスにロードすること



ELT: 抽出したデータをデータ ウェアハウスにロードしてその中で変換処理をすること



# ワークフローの選び方

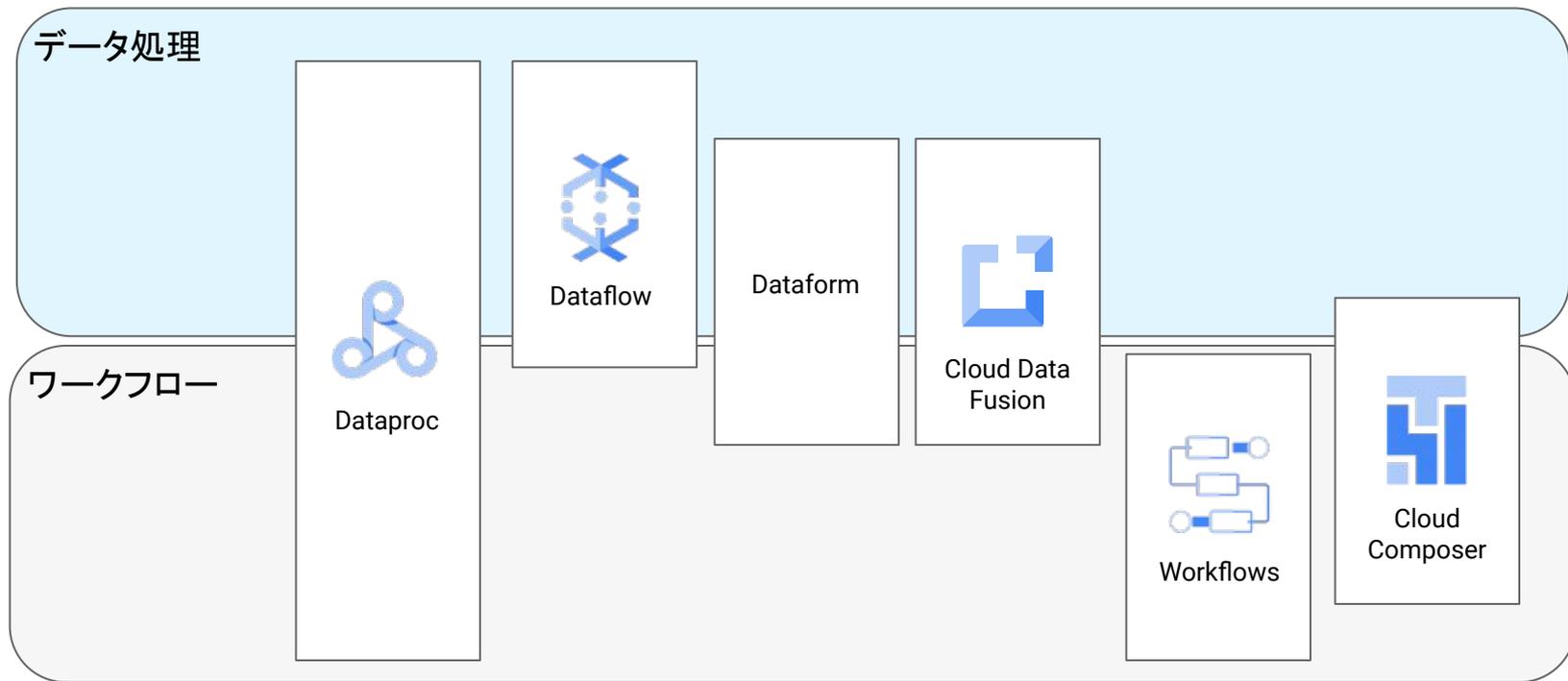
- 処理の実行順序の柔軟性(順次実行、分岐等)
- スケジュール機能
- 連携可能なサービスの種類
- エラー処理
  - リトライ
  - 通知
  - 途中からの実行
  - 幂等
- 開発方法
- ワークフローの監視
- 環境の管理

Google Cloud の  
データパイプライン  
サービス



# データパイプライン サービス

- Google Cloud のサービスをデータ処理、ワークフローの観点で分けると



# ご紹介する

## Google Cloud のワークフロー機能を含むサービス

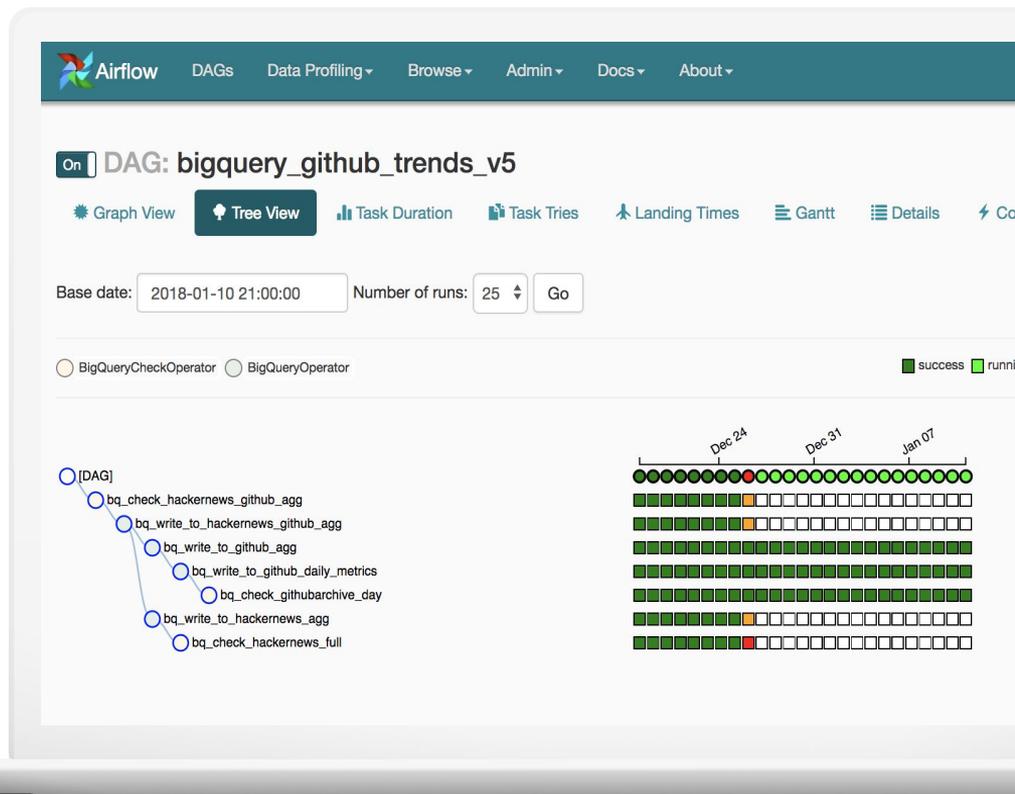
- Cloud Composer
- Workflows
- Cloud Data Fusion
- Dataform (preview)

# サービスの比較

	 Cloud Composer	 Workflows	 Cloud Data Fusion	Dataform
ユースケース	複数のデータ処理サービスを束ねるワークフロー エンジンとしての利用 複雑なワークフローやエラー ハンドリングなど	API ベースのサービスを束ねるワークフローエンジンとしての利用。複雑な処理の自動化	データ統合のための変換処理としての利用がメイン	DWH 内での依存関係をもった変換処理をバッチとして実行
運用・管理	フルマネージド	サーバレス	フルマネージド	サーバレス
依存関係の定義	タスクの順次実行だけでなく分岐など柔軟な定義が可能	タスク(ステップ)の順次実行だけでなく分岐など柔軟な定義が可能	変換処理(パイプライン ノード)の流れの依存関係定義がメイン。	変換処理(アクション)の順次実行定義が可能
スケジュール機能	実行間隔だけでなく開始日、終了日などの定義でより柔軟なスケジューリングが可能	Cloud Scheduler(フルマネージドのcron サービス) で定義	cron ベースでの定義 (GUI で定義可能)	cron ベースでの定義 (GUI で定義可能)
開発言語	Python でワークフローの定義	YAML でワークフローの定義	GUI でパイプラインの定義 ビルトインの変換処理も多数提供されている	主に SQL で JavaScript やビルトイン関数などを合わせてパイプラインの定義
連携可能なサービス	様々な種類のサービスと連携できるようになっている。(Operatorとして提供) カスタム Operator を作成可能	googleapi と http	多数のサービスとの連携が可能。(プラグインとして提供)カスタムプラグインの作成も可能	特定の DWH 内での処理のみ
エラー処理	タスクのリトライの柔軟な設定が可能。失敗したワークフローの再実行、途中のタスクからの実行など柔軟な再実行が可能	ステップのリトライの柔軟な設定が可能。失敗したワークフローの再実行というよりは新たに実行しなおす。	変換処理のリトライはビルトインの種類による。失敗したパイプラインは、再実行というよりは新たに実行しなおす。	リトライ処理はなし。失敗したパイプラインは再実行というよりは新たに実行しなおす。パイプラインの途中からの実行は可能
エラー通知	email 通知やエラー時の callback の定義で通知可能	単体ではエラー通知の機能はない	email、http コールでの通知	email、slack での通知
監視	様々な角度から監視できるように多数の UI を提供	実行状態の監視は明示的にログ出力が必要	実行状態をログで監視可能 変換の input output の件数も監視可能	UI のログから実行状態を監視可能

# Cloud Composer

- Apache Airflow のマネージド サービス
- ワンクリックで Apache Airflow の実行環境の構築、および、ワークフローのデプロイが実施可能
- Python でワークフローを記述
- 各 Google Cloud サービス (BigQuery、Cloud Dataflow、Cloud Dataproc など) と連携するための パッケージが  
プリ インストール



# Cloud Composer の開発

- Python でワークフロー(DAG)を開発
- Operator で各タスクを定義
  - 多数のビルトイン operator
  - カスタムで開発も可能
- Airflow の dags フォルダに Python のコードを配置

```
default_dag_args = {
    # The start_date describes when a DAG is valid / can be run. Set this to a
    # fixed point in time rather than dynamically, since it is evaluated every
    # time a DAG is parsed. See:
    # https://airflow.apache.org/faq.html#what-s-the-deal-with-start-date
    'start_date': datetime.datetime(2018, 1, 1),
}

# Define a DAG (directed acyclic graph) of tasks.
# Any task you create within the context manager is automatically added to the
# DAG object.
with models.DAG(
    'composer_sample_simple_greeting',
    schedule_interval=datetime.timedelta(days=1),
    default_args=default_dag_args) as dag:
    # [END composer_simple_define_dag]
    # [START composer_simple_operators]
    def greeting():
        import logging
        logging.info('Hello World!')

    # An instance of an operator is called a task. In this case, the
    # hello_python task calls the "greeting" Python function.
    hello_python = python_operator.PythonOperator(
        task_id='hello',
        python_callable=greeting)

    # Likewise, the goodbye_bash task calls a Bash script.
    goodbye_bash = bash_operator.BashOperator(
        task_id='bye',
        bash_command='echo Goodbye. ')
    # [END composer_simple_operators]

    # [START composer_simple_relationships]
    # Define the order in which the tasks complete by using the >> and <<
    # operators. In this example, hello_python executes before goodbye_bash.
    hello_python >> goodbye_bash
    # [END composer_simple_relationships]
# [END composer_simple]
```

# Cloud Composer の依存関係の定義

- Operator を >> でつなぐことで順次実行を定義
- ブランチ化や条件分岐などの複雑な処理も定義可能

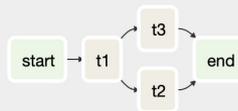


```
with models.DAG(...
) as dag:

    start = DummyOperator(task_id="start")
    end = DummyOperator(task_id="end")

    t1 = BashOperator(
        task_id="t1",
        bash_command="echo t1",
    )
    t2 = BashOperator(
        task_id="t2",
        bash_command="echo t2 "
    )

    start >> t1 >> t2 >> end
```



```
with models.DAG(...
) as dag:

    start = DummyOperator(task_id="start")
    end = DummyOperator(task_id="end")

    t1 = BashOperator(
        task_id="t1",
        bash_command="echo t1",
    )
    t2 = BashOperator(
        task_id="t2",
        bash_command="echo t2 "
    )
    t3 = BashOperator(
        task_id="t3",
        bash_command="echo t3 "
    )

    start >> t1 >> [t2,t3] >> end
```

# Cloud Composer の連携可能なサービス operators

- 3つのタイプの Operators
  - Action: 処理の実行、または外部システムに処理の実行を依頼 (例: dataflowジョブの起動)
  - Transfer: データの移動
  - Sensors: 特定の基準が満たされるまで実行 (例: ファイル検知)

## Azure: Microsoft Azure

Logging

Azure Blob Storage

Azure File Share

Azure CosmosDB

Azure Data Lake

Azure Container Instances

## AWS: Amazon Web Services

Logging

AWS EMR

AWS S3

AWS Batch Service

AWS RedShift

AWS DynamoDB

AWS Lambda

AWS Kinesis

Amazon SageMaker

## GCP: Google Cloud Platform

Logging

GoogleCloudBaseHook

BigQuery

Cloud Spanner

Cloud SQL

Cloud Bigtable

Cloud Build

Compute Engine

## Cloud Functions

Cloud DataFlow

Cloud DataProc

Cloud Datastore

Cloud ML Engine

Cloud Storage

Transfer Service

Cloud Vision

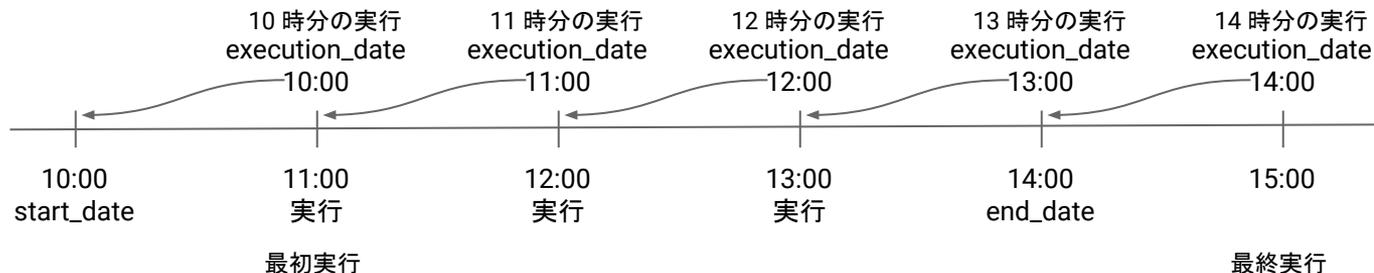
Cloud Text to Speech

Cloud Speech to Text

Cloud Speech Translate Operators

# Cloud Composer のスケジュール機能

- cron ベースでの定義
  - プリセット: None、@once、@hourly、@daily、@weekly、@monthly、@yearly
  - cron 形式の定義
- 開始日、終了日、実行間隔を定義することで DAG の実行をアトミックで冪等に
  - 各 DAG 実行にいつ分の実行かを示す execution\_date が割り当てられる。
  - 同一 execution\_date は 1 つのみ
  - 再実行しても execution\_date は変わらない
  - backfill、catchup の実行も可能



# Cloud Composer のエラー処理

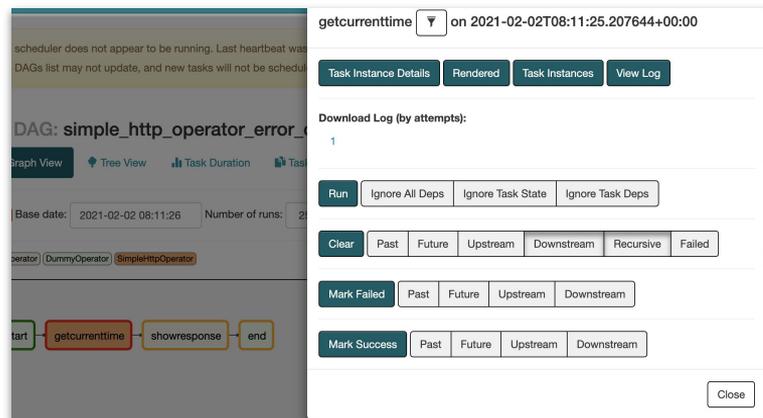
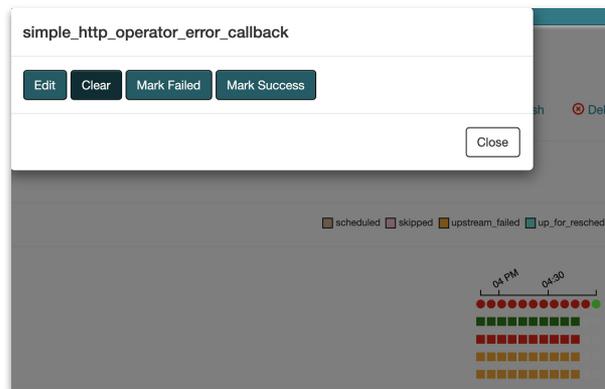
- 通知
  - 失敗時、リトライ時に email 送信可能(メールサービスは自前で要準備)
  - 失敗時、リトライ時、成功時に callback が指定できるので email 以外のカスタマイズした通知も可能

```
default_args = {  
    "start_date": days_ago(1),  
    "email": ["te****@google.com"],  
    "email_on_failure": True,  
    "email_on_retry": False  
}
```

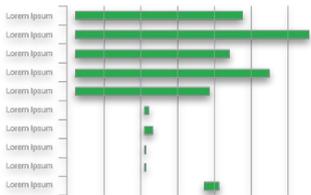
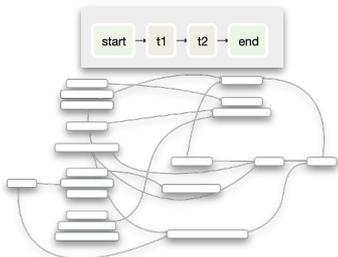
```
def send_message(context):  
    通知処理  
  
default_args = {  
    "start_date": days_ago(1),  
    "on_failure_callback": send_message  
}
```

# Cloud Composer のエラー処理

- 実行中のタスクに対するリトライ設定可能
  - リトライ回数、間隔
- 失敗した DAG の再実行
  - DAG のステータスをクリアで、同一の execution\_date で再実行
  - 途中のタスクからのレジューム実行も可能



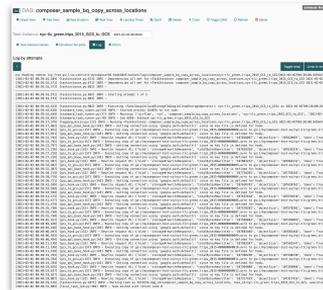
# Cloud Composer の監視



**Tree views** 遅いパイプラインの**ブロッカー**を特定するのに役立つ

**Graph views** ワークフローの**依存関係**とステータスを包括的に確認可能

**Gantt views** タスクの継続時間を分析してボトルネックを特定可能



ログを参照可能

# Workflows

- サーバレス
- 迅速なスケラビリティ
- 複雑な処理の自動化
- 組み込みのエラー処理
- Google Cloud 利用時の組み込み認証
- http ベースのサービス連携

ワークフロー ← ワークフローの作成 SYNTAX REFERENCE 可視化 >

構成 1 ワークフローの定義 2

1 ワークフローの構文を使用してワークフローを定義します。 [構文リファレンス](#)と[サンプルワークフロー](#)をご覧ください。  
閉じる

```
1 # This is a sample workflow, feel free to replace it with your source code
2 #
3 # This workflow does the following:
4 # - reads current time and date information from an external API and stores
5 #   the response in CurrentDateTime variable
6 # - retrieves a list of Wikipedia articles related to the day of the week
7 #   from CurrentDateTime
8 # - returns the list of articles as an output of the workflow
9
10 - getCurrentTime:
11   call: http.get
12   args:
13     url: https://us-central1-workflowsample.cloudfunctions.net/datetime
14   result: CurrentDateTime
15 - readWikipedia:
16   call: http.get
17   args:
18     url: https://en.wikipedia.org/w/api.php
19     query:
20       action: opensearch
21       search: ${CurrentDateTime.body.dayOfTheWeek}
22   result: WikiResult
23 - returnOutput:
24   return: ${WikiResult.body[1]}
```

MAIN

```
graph TD
  START((START)) --> A[getCurrentTime  
call]
  A --> B[readWikipedia  
call]
  B --> C[returnOutput  
return]
  C --> END((END))
```

Google Cloud

# Workflows の開発

- YAML でワークフローを定義
- Console での開発も可能
- 処理の流れを可視化

```
# 各処理をステップと呼ぶ
# getMessage と returnValue がステップ
- getMessage:
  call: http.post
  args:
    url: https://www.example.com/endpoint
    body:
      someVal: "Hello World"
      anotherVal: 123
  result: theMessage
- returnValue:
  return: ${theMessage.body}
```

The screenshot displays the Google Cloud Workflows console interface. The left pane shows the workflow definition in YAML format, and the right pane shows a visual flowchart of the workflow.

**Workflow Definition (YAML):**

```
1 # This is a sample workflow, feel free to replace it with your source code
2 #
3 # This workflow does the following:
4 # - reads current time and date information from an external API and stores
5 #   the response in CurrentDateTime variable
6 # - retrieves a list of Wikipedia articles related to the day of the week
7 #   from CurrentDateTime
8 # - returns the list of articles as an output of the workflow
9
10 - step1:
11   assign:
12     - a: 1
13 - step2:
14   switch:
15     - condition: ${a==1}
16     steps:
17       - stepA:
18         assign:
19           - a: ${a+7}
20       - stepB:
21         return: ${"increase a to:"+string(a)}
22 - step3:
23   return: ${"default a="+string(a)}
```

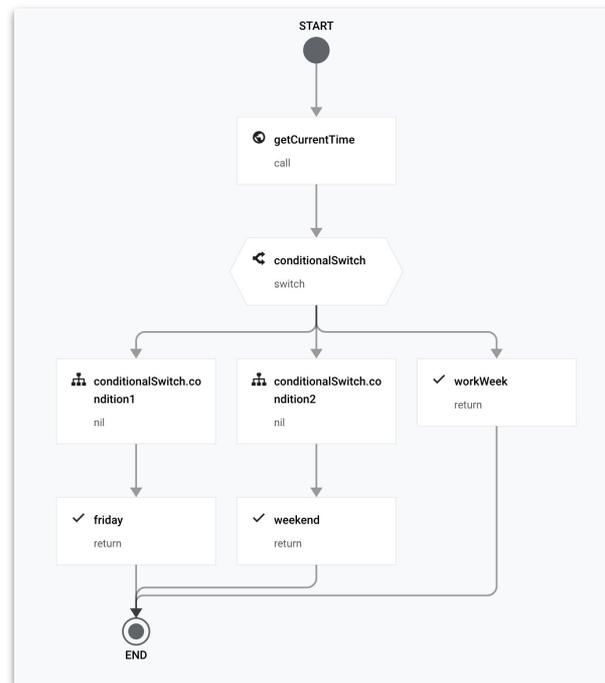
**Visualization (Flowchart):**

- START node leads to step1 (assign).
- step1 leads to step2 (switch).
- step2 branches into two paths based on the condition `step2.condition`.
- The first path goes through stepA (assign) and then stepB (return).
- The second path goes through step3 (return).
- Both paths converge at the END node.

# Workflows の依存関係の定義

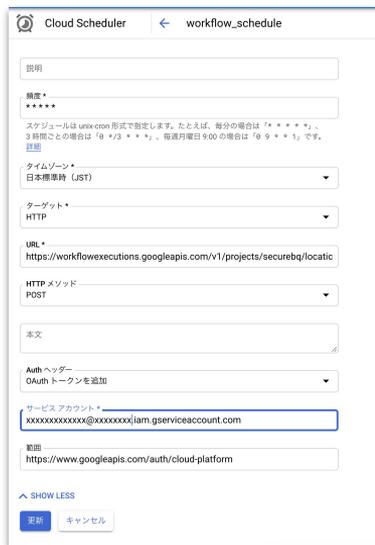
- 基本的には書いた順序に実行される
- 複雑な処理も定義可能
  - 制御文で条件分岐
  - サブワークフローを定義し実行

```
- getCurrentTime:  
  call: http.get  
  args:  
    url: https://us-central1-workflowsample.cloudfunctions.net/datetime  
  result: currentTime  
- conditionalSwitch:  
  switch:  
    - condition: ${currentTime.body.dayOfTheWeek == "Friday"}  
      next: friday  
    - condition: ${currentTime.body.dayOfTheWeek == "Saturday" OR  
      currentTime.body.dayOfTheWeek == "Sunday"}  
      next: weekend  
  next: workWeek  
- friday:  
  return: "It's Friday! Almost the weekend!"  
- weekend:  
  return: "It's the weekend!"  
- workWeek:  
  return: "It's the work week."
```



# Workflows のスケジュール機能

- Workflows 自体にスケジュール機能はなく cloud scheduler で設定する
- cron 形式で定義



The screenshot shows the Cloud Scheduler configuration page for a job named 'workflow\_schedule'. The form includes the following fields:

- 説明**: Empty text input field.
- 指定 \***: Empty cron expression input field.
- タイムゾーン \***: Dropdown menu set to '日本標準時 (JST)'. A note below explains that the scheduler uses Unix cron format and provides examples for daily, every 3 days, and monthly schedules.
- ターゲット \***: Dropdown menu set to 'HTTP'.
- URL \***: Text input field containing 'https://workflowexecutions.googleapis.com/v1/projects/securebg/locati'.
- HTTP メソッド**: Dropdown menu set to 'POST'.
- 本文**: Empty text input field.
- Auth ヘッダー**: Dropdown menu set to 'OAuth トークンを追加'.
- サービス アカウント \***: Text input field containing 'xxxxxxxxxxxx@xxxxxxxxx.iam.gserviceaccount.com'.
- 範囲**: Text input field containing 'https://www.googleapis.com/auth/cloud-platform'.

At the bottom, there are 'SHOW LESS' and '戻る' (Back) buttons, and a 'キャンセル' (Cancel) button.

```
gcloud scheduler jobs create http JOB_NAME \  
  --schedule="*/5 * * * *" \  
  --uri="https://workflowexecutions.googleapis.com/v1/projects/PROJECT_NAME/locations/REGION_\  
  NAME/workflows/WORKFLOW_NAME/executions" \  
  --time-zone="TIME_ZONE" \  
  --oauth-service-account-email="SERVICE_ACCOUNT_NAME@PROJECT_NAME.iam.gserviceaccount.com"
```

# Workflows が連携可能なサービス

- http ベースの API サービス
- Google Cloud API (Pub/Sub、Firestore 等)

```
- STEP_NAME:
  call: {http.get|http.post|http.request}
  args:
    url: URL_VALUE
    [method: REQUEST_METHOD]
    [headers:
      KEY:VALUE
      ...]
    [body:
      KEY:VALUE
      ...]
    [query:
      KEY:VALUE
      ...]
    [auth:
      type:{OIDC|OAuth2}]
    [timeout: VALUE_IN_SECONDS]
  [result: RESPONSE_VALUE]
```

```
- callMyFunction:
  call: http.post
  args:
    url:
      https://us-central1-myproject123.cloudfunctions.net/myfunc1
    auth:
      type: OIDC
    body:
      someVal: "Hello World"
      anotherVal: 123
  result: theMessage
- returnValue:
  return: ${theMessage.body}
```

# Workflows エラー処理

- 組み込みの通知はないので例外ハンドリングやカスタムのロギングで
- 実行中のステップに対するリトライは可能
- ステップの例外ハンドリングも可能
- 失敗したワークフローの再実行
  - 再実行というよりは新規に実行
  - 途中のステップからの実行は不可

```
- step_name:  
  try:  
    steps:  
      ...  
  retry:  
    predicate: ${http.default_retry_predicate_non_idempotent}  
    max_retries: 10  
    backoff:  
      initial_delay: 1  
      max_delay: 90  
      multiplier: 3
```

実行数	ログ	詳細	ソース		
≡ 実行をフィルタ					
状態	実行 ID	ワークフローのリビジョン	開始時間 ↓	終了時間	期間
❌ 失敗	87da4df8-6e58-4c8d-8ab3-168803e91ad4	000002-e8c	2021/01/28 23:58:08	2021/01/28 23:58:09	0.497 秒
✅ 完了	3bdb673-0a70-4b94-a396-a90ff2337ee8	000001-441	2021/01/28 23:57:08	2021/01/28 23:57:08	0.391 秒

```
- read_item:  
  try:  
    call: http.get  
    args:  
      url: https://host.com/api  
    result: api_response  
  except:  
    as: e  
    steps:  
      - known_errors:  
        switch:  
          - condition: ${not("HttpError" in e.tags)}  
            return: "Connection problem."  
          - condition: ${e.code == 404}  
            return: "Sorry, URL wasn't found."  
          - condition: ${e.code == 403}  
            return: "Authentication error."  
      - unhandled_exception:  
        raise: ${e}  
- url_found:  
  return: ${api_response.body}
```

# Workflows の監視

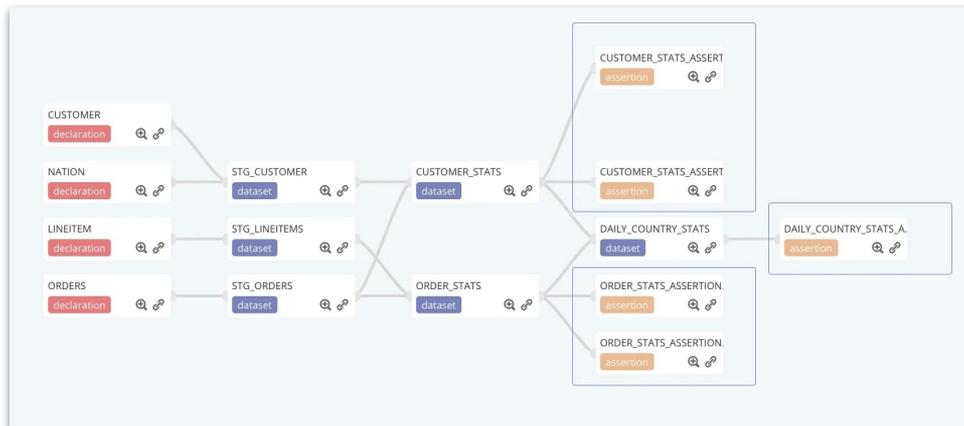
- デフォルトでは実行状態はロギングされない
- Cloud logging との連携で監視
  - カスタム ロギングで実行状況を出力して監視

```
- step1:  
  assign:  
    - varA: "Hello"  
    - varB: "World"  
- logStep:  
  call: sys.log  
  args:  
    text: TEXT  
    severity: SEVERITY  
- step2:  
  return: ${varA + " " + varB}
```

# Dataform

- DWH 内でデータパイプラインを構築する ELT の T(変換) に特化
- SQL が書ける Analyst がセルフサービスでデータの準備が可能
- DWH の処理能力、拡張性、可用性などのメリットを享受
- バージョン管理、テスト、コラボレーションの仕組み

```
stg_customer.sqlx x
1  config {
2    | type: "table"
3  }
4
5  select r_name,
6         | | | | n_name,
7         | | | | cust.*
8  from
9  ${ref("stg_nation")} as nation , ${ref("customer")} as cust
10 where n_nationkey = c_nationkey
11
```



# Dataform で利用可能なデータウェアハウス

- 外部サービスとの連携は不可
- ワークフローを定義、実行可能な対応データウェアハウス
  - BigQuery
  - Amazon Redshift
  - Snowflake
  - Azure SQL Data Warehouse
  - PostgreSQL

[Supported warehouses](#)

# Dataform の開発方法

- SQLX という SQL の拡張言語で開発
  - データ変換処理部分は、基本的に SQL による記述
  - 拡張部分：メタデータ、依存テーブルの指定などのビルトイン関数、JavaScript による動的な SQL の生成

```
config {  
  type: "table",  
  description: "combine natio table and region table",  
  columns: {  
    r_name: "region name",  
    n_name: "country name",  
    n_nationkey: "country id"  
  }  
}  
  
select r_name,  
       n_name,  
       n_nationkey  
from  
${ref("nation")} , ${ref("region")}  
where r_regionkey = n_regionkey
```

# Dataform 依存関係の定義

- ビルトイン関数 `ref("参照先")` で定義

```
config {
  type: "table",
  description: "customer table joined with customer and
stg_nation",
  columns: {
    r_name: "region name",
    n_name: "nation name",
    cust: "all customer columns"
  }
}

select r_name,
       n_name,
       cust.*
from
  ref("stg_nation") as nation , ref("customer") as cust
where n_nationkey = c_nationkey
```



# Dataform スケジュール定義

- cron ベース
  - cron 定義でも GUI でも定義可能
  - タイムゾーンは UTC のみ

environments.json x

ENVIRONMENTS > PRODUCTION > NEW\_SCHEDULE

**new\_schedule (schedule)**

Schedule name

new\_schedule

Schedule

Dataform runs your project at a schedule configured with a cron expression. Use the options below to create a schedule.

Running every day at 11:58

Current UTC time: 02:58

Repeats

CHOOSE FREQUENCY

SELECT SCHEDULE TYPE

CANCEL SAVE

OR

58 11 \* \* \*

For help creating a custom cron spec, go here

Repeats

DAILY

At

11 : 58

The above time will be interpreted as UTC. Current UTC time: 02:59

On days

Monday

Tuesday

Wednesday

Thursday

Friday

Saturday

Sunday

CANCEL SAVE

定義は json ファイルに追加される

```
1 {
2   "environments": [
3     {
4       "name": "production",
5       "configOverride": {},
6       "schedules": [
7         {
8           "name": "new_schedule",
9           "cron": "0 3 * * mon,tue,wed,thu,fri,sat,sun"
10        }
11      ],
12      "gitRef": "master"
13    }
14  ]
15 }
```

# Dataform エラー処理

- 通知
  - email or slack が選べる
  - schedule に対しての紐付け
  - 成功時にも通知可

NOTIFICATION CHANNELS > NEW\_NOTIFICATION\_CHANNEL

**Notification channel name**  
Give the new notification channel a distinctive name so that it can be reused in other schedules.

Email  Slack

**List of email addresses**  
Each email address will receive a notification when the schedule finishes.

ENVIRONMENTS > PRODUCTION > 5MINSCHEDULE

**5minschedule (schedule)**

**Schedule name**

**Schedule**  
Dataform runs your project at a schedule configured with a cron expression. Use the options below to c

Running every 5 minutes (UTC)  
Current UTC time: 10:55

OR

For help creating a custom cron spec, go here

**Enable this schedule**

**Run configuration**

**Tags to run**  
If left empty, the whole project (2 actions) will run.

**Run with full refresh**  
If enabled, incremental datasets will be deleted and then rebuilt from scratch.

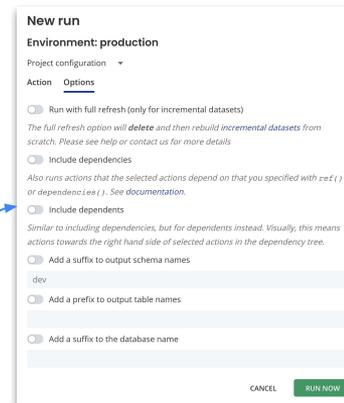
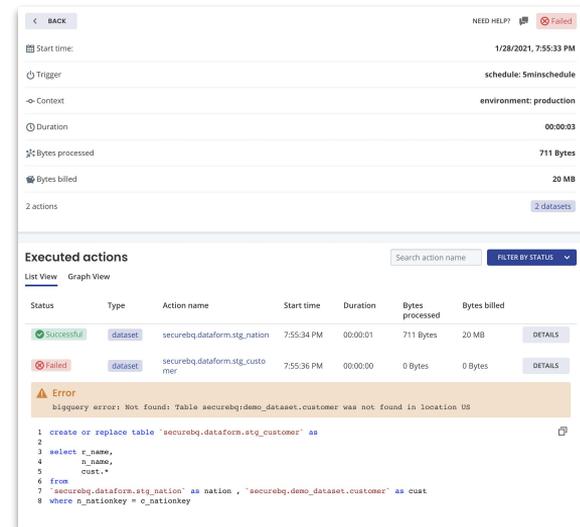
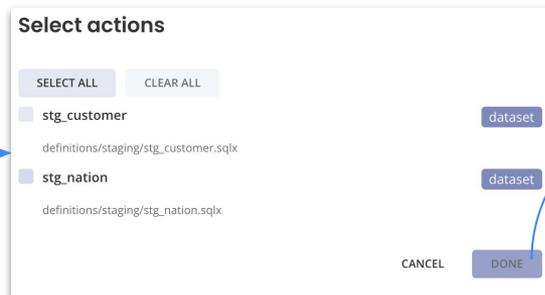
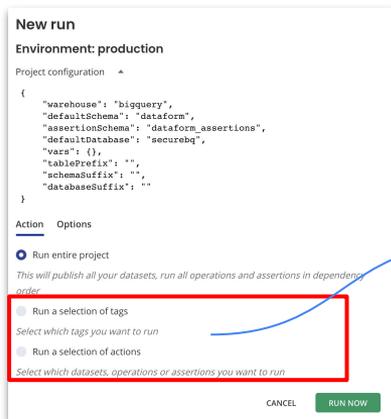
**Include dependencies**  
If enabled, any dependencies of the schedule's selected actions will also run.

**Include dependents**  
If enabled, any dependents of the schedule's selected actions will also run.

**Notification settings**  
Notify  upon

# Dataform エラー処理

- 実行中のアクション (各 SQL の処理) のリトライはなし
- 失敗したワークフローの再実行
  - 再実行というよりは新規に実行
  - 途中からの実行は可能
    - アクションやタグ単位での実行ができる
    - 依存関係のある処理もあわせて実行できる



# Dataform の監視

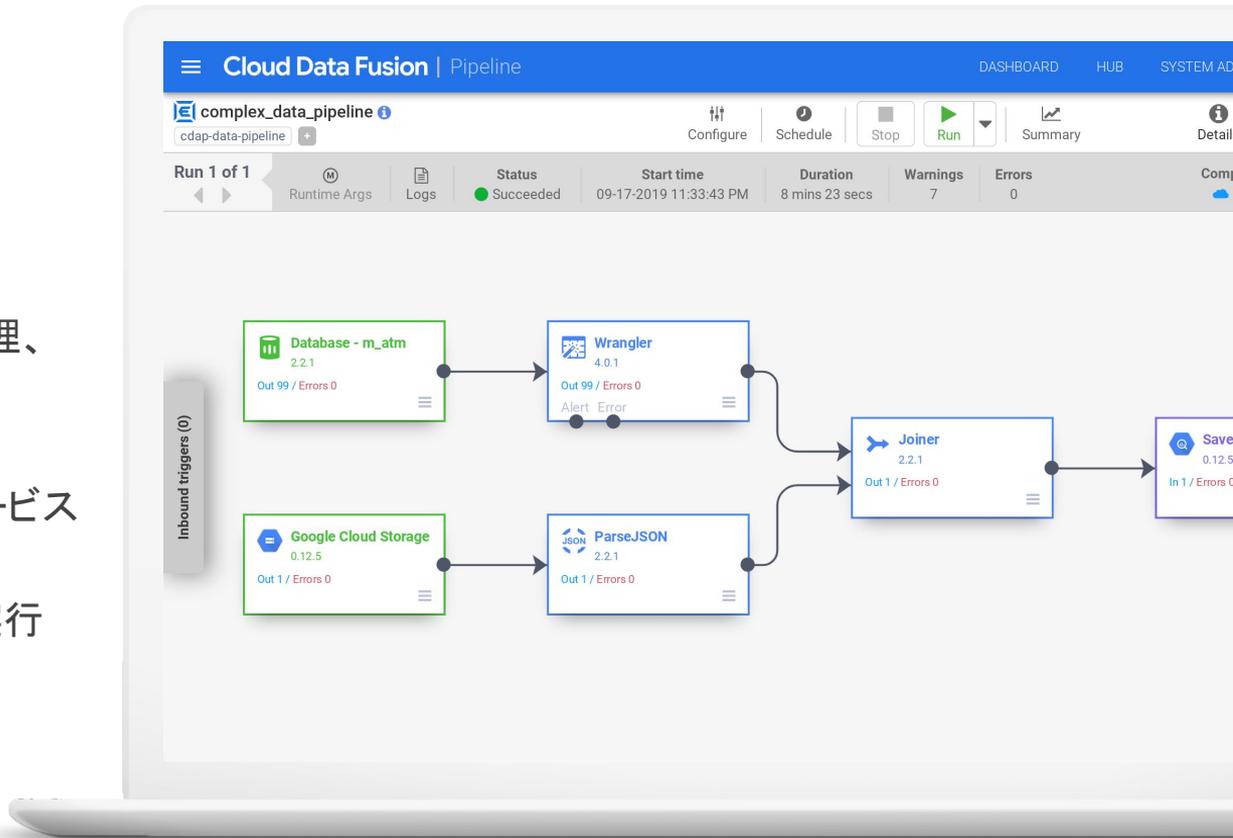
- Dataform UI のログから実行状態を監視することが可能

The screenshot shows a Dataform job in progress. The top summary includes: Start time: 2/1/2021, 2:26:20 AM; Trigger: manual; Context: branch: tenishim\_dev; Duration: 00:00:48; Bytes processed: 2.42 GB; Bytes billed: 2.44 GB; 3 actions. The 'Executed actions' section is in 'List View' mode. A table with columns Status, Type, Action name, Start time, Duration, Bytes processed, and Bytes billed is shown. The first row is 'Successful', 'dataset', 'securebq.dataform.stg\_nation', '2:26:23 AM', '00:00:02', '711 Bytes', '20 MB'. Below this, the SQL code for the 'stg\_nation' table is displayed. The second row is 'Running', 'dataset', 'securebq.dataform.stg\_customer', '2:26:25 AM', '2.42 GB', '2.42 GB'. Below this, the SQL code for the 'stg\_customer' table is displayed, with the 'from' clause and the table names 'stg\_nation' and 'stg\_customer' highlighted by a red box. A blue arrow points from this red box to the 'Graph View' of the 'Executed actions' section in the adjacent screenshot.

The screenshot shows a Dataform job that has completed successfully. The top summary includes: Start time: 2/1/2021, 2:26:20 AM; Trigger: manual; Context: branch: tenishim\_dev; Duration: 00:02:21; Bytes processed: 101.78 GB; Bytes billed: 101.8 GB; 3 actions. The 'Executed actions' section is in 'Graph View' mode. It displays a pipeline graph with three nodes: 'stg\_nation' (Successful), 'stg\_customer' (Successful), and 'customer\_orders' (Running). A blue arrow points from the 'Graph View' tab in the left screenshot to this graph.

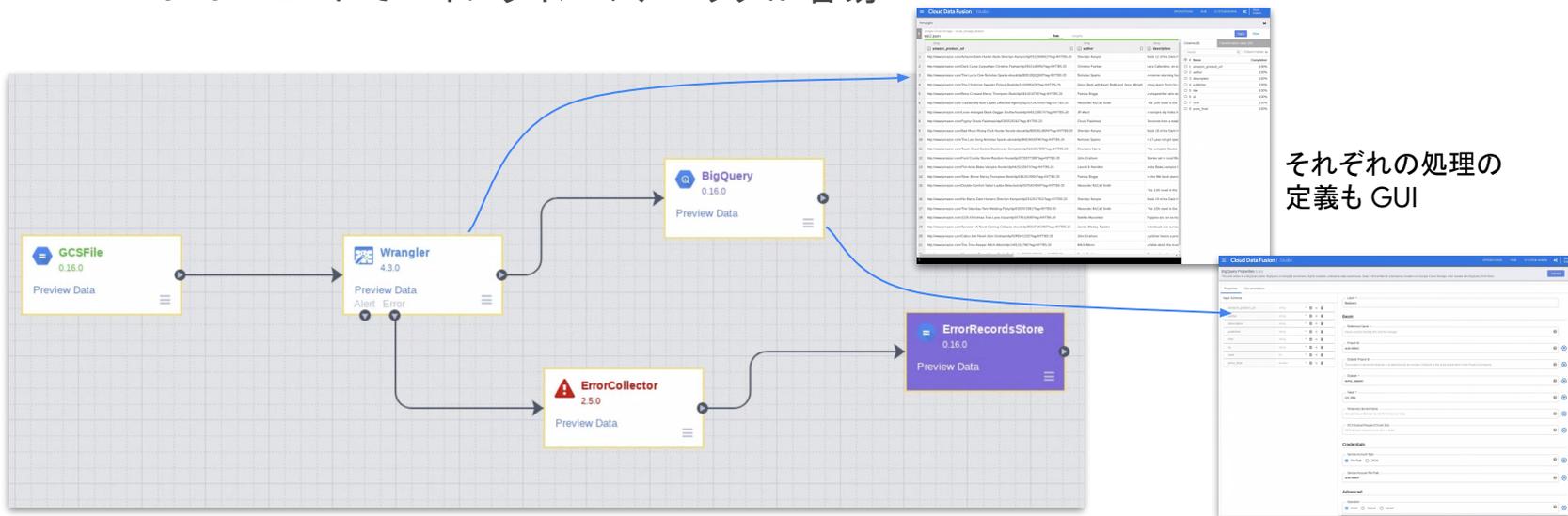
# Cloud Data Fusion

- フルマネージドのデータ統合サービス
- OSS の CDAP がベース
- GUI 操作でデータの収集、処理、整形、加工を実施可能
- ストリーミングとバッチ対応
- Google Cloud 含め多くのサービスとの連携が可能
- パイプラインは Dataproc で実行



# Cloud Data Fusion の開発

- GUI で開発
  - データソース、変換、シンク等のプラグインをつなげてパイプラインを構築
  - コーディング可能なプラグインもある
- Preview モードでパイプラインのデバッグが容易





# Cloud Data Fusion の連携可能なサービス プラグイン

- データソース、データシンク、アクション プラグインで多くの外部サービスと連携
- その他、汎用的な変換、分析、エラー処理用など様々なプラグインを提供

データソース		データシンク		アクション (コマンドを実行)	
Oracle	Database (JDBC)	Oracle	Database (JDBC)	Oracle Execute	Netezza Execute
SQL Server	GCS	SQL Server	GCS	IBM DB2 Execute	Snowflake Run SQL
IBM DB2	S3	IBM DB2	S3	SQL Server Execute	Snowflake to Cloud
MySQL	Azure Blob Store	MySQL	Salesforce	MySQL Execute	Storage
PostgreSQL	Salesforce	PostgreSQL	Send Grid	PostgreSQL	Cloud Storage to
Redshift	ServiceNow	Redshift	Elasticsearch	Execute	Snowflake
Teradata	Elasticsearch	Teradata	Splunk	Redshift to S3	Database (JDBC)
Netezza	Splunk	Netezza	Cloud Spanner	S3 to Redshift	S3 Client
Snowflake	Cloud Spanner	Snowflake	その他	Teradata Execute	その他
BigQuery	その他	BigQuery			

# Cloud Data Fusion スケジュール機能

- cron ベース
  - cron 形式でも GUI 定義でも可能
- パイプラインの同時実行も制御可能

Configure schedule for pipeline DataFusionQuickstart2

**Basic** | **Advanced**

Pipeline run repeats

- Every 5 min
- Every 10 min**
- Every 30 min
- Hourly
- Daily
- Weekly
- Monthly
- Yearly

Summary  The pipeline cannot have concurrent runs.

Max concurrent runs

Compute profiles

Configure schedule for pipeline DataFusionQuickstart2 ✕

**Basic** | **Advanced**

Schedule this pipeline by using Cron syntax

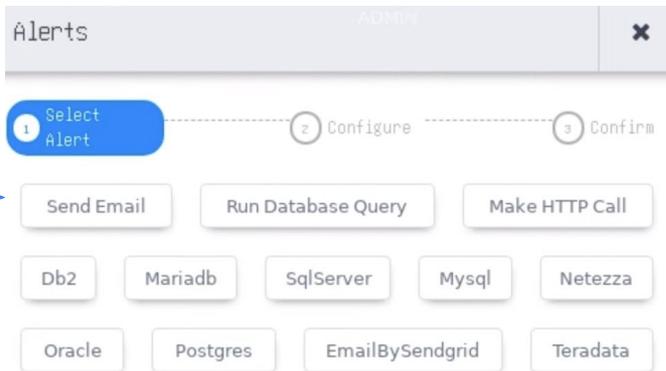
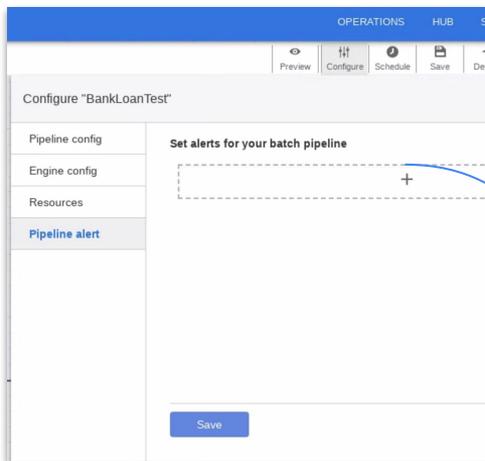
Min Hour Day Month Days of the week

Max concurrent runs

Compute profiles

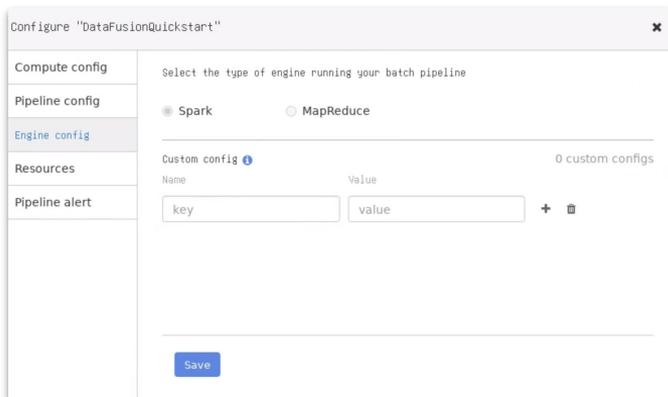
# Cloud DataFusion エラー処理

- 通知処理
  - Email や HTTP call などのアラートがパイプラインの completion, failure, success の状態から選択可能
  - JDBC でデータベースにつないでクエリの実行も可能



# Cloud Data Fusion エラー処理

- 実行中のノードのリトライは可能
  - Spark や MapReduce の設定を利用可能
  - Data fusionから設定を上書きすることが可能
- 失敗したパイプラインの再実行
  - 再実行というよりは新規での実行
  - 途中のノードからの実行は不可



The screenshot shows a configuration window titled "Configure 'DataFusionQuickstart'". On the left is a sidebar with menu items: "Compute config", "Pipeline config", "Engine config" (highlighted), "Resources", and "Pipeline alert". The main content area is titled "Select the type of engine running your batch pipeline" and contains two radio buttons: "Spark" (selected) and "MapReduce". Below this is a section for "Custom config" with a sub-header "0 custom configs". It features a table with columns "Name" and "Value". One row is visible with "key" in the Name column and "value" in the Value column. There are "+" and "-" icons to the right of the table. At the bottom of the configuration area is a blue "Save" button.

# Cloud Data Fusion の監視

- 実行中のデータの処理状況を確認可能
  - Input / Output 件数
- ログもリアルタイムで監視可能



Cloud Data Fusion | Pipeline

BankLoanTest

cdap-pipeline

Run 3 of 3

Time	Level	Message
02/01/2021 14:43:21	INFO	Creating Dataproct cluster cdap-bankloan-d3e3eb29-649b-11eb-b54f-0aa20c308fca in project securebp02, in region us-central1, with image 1.3, with system labels {goog-datafusion-version=6_2, cdap-version=6_3_0-1610599233779, goog-datafusion-edition=basic}
02/01/2021 14:46:41	INFO	Starting Workflow Program 'DataPipelineWorkflow' with Arguments [logical start time=1612190660257, system profile name=SYSTEM dataproct], with debugging false
02/01/2021 14:46:41	INFO	JVM properties {logback.configurationFile=logback.xml, twill.container.class.loader=io.cdap.cdap.common.app.MainClassLoader}
02/01/2021 14:46:41	INFO	Starting runnable DataPipelineWorkflow for runid program_run_default:BankLoanTest-SNAPSHOT workflow DataPipelineWorkflow d3e3eb29-649b-11eb-b54f-0aa20c308fca with job manager.
02/01/2021 14:46:53	INFO	Twill program running: program_run_default:BankLoanTest-SNAPSHOT workflow DataPipelineWorkflow d3e3eb29-649b-11eb-b54f-0aa20c308fca, twill runid: d3e3eb29-649b-11eb-b54f-0aa20c308fca
02/01/2021 14:47:17	INFO	Messaging metadata table created at data/messaging/namespace system.tms.meta
02/01/2021 14:47:19	INFO	Log Appender stackdriver is initialized and started.
02/01/2021 14:47:20	INFO	Core Messaging Service started
02/01/2021 14:47:20	INFO	Starting HTTP Service messaging.service at address cdap-bankloan-d3e3eb29-649b-11eb-b54f-0aa20c308fca-us-central1-c-securebp02.internal/10.11.0.58.0
02/01/2021 14:47:20	INFO	Messaging HTTP server started on cdap-bankloan-d3e3eb29-649b-11eb-b54f-0aa20c308fca-m-us-central1-c-securebp02.internal/10.11.0.58.35643
02/01/2021 14:47:24	WARN	file /tmp/default_BankLoanTest_DataPipelineWorkflow_d3e3eb29-649b-11eb-b54f-0aa20c308fca/cConf.xml an attempt to override final parameter: messaging.system.topics: Ignoring
02/01/2021 14:47:24	WARN	file /tmp/default_BankLoanTest_DataPipelineWorkflow_d3e3eb29-649b-11eb-b54f-0aa20c308fca/cConf.xml an attempt to override final parameter: app.program.runtime.monitor.topics.configs: Ignoring
02/01/2021 14:47:24	WARN	file /tmp/default_BankLoanTest_DataPipelineWorkflow_d3e3eb29-649b-11eb-b54f-0aa20c308fca/cConf.xml an attempt to override final parameter: messaging.max.instances: Ignoring
02/01/2021 14:47:24	WARN	file /tmp/default_BankLoanTest_DataPipelineWorkflow_d3e3eb29-649b-11eb-b54f-0aa20c308fca/cConf.xml an attempt to override final parameter: app.program.runtime.monitor.server.info.file: Ignoring
02/01/2021 14:47:33	INFO	Starting Workflow Program 'DataPipelineWorkflow' with Arguments [logical start time=1612190660257, system profile name=SYSTEM dataproct], with debugging false
02/01/2021 14:47:47	INFO	Application workflow default:BankLoanTest:DataPipelineWorkflow with id application_1612190660504_0001 submitted
02/01/2021 14:47:57	INFO	Yarn application workflow default:BankLoanTest:DataPipelineWorkflow application_1612190660504_0001 is in state RUNNING
02/01/2021 14:47:58	INFO	Twill program running: program_run_default:BankLoanTest-SNAPSHOT workflow DataPipelineWorkflow d3e3eb29-649b-11eb-b54f-0aa20c308fca, twill runid: 2629f742-9725-4e7e-9475-8bc4b4502b9
02/01/2021 14:48:31	INFO	Runnable individual:DataPipelineWorkflow

# まとめ

	 Cloud Composer	 Workflows	 Cloud Data Fusion	Dataform
ユースケース	複数のデータ処理サービスを束ねるワークフロー エンジンとしての利用 複雑なワークフローやエラー ハンドリングなど	API ベースのサービスを束ねるワークフローエンジンとしての利用。複雑な処理の自動化	データ統合のための変換処理としての利用がメイン	DWH 内での依存関係をもった変換処理をバッチとして実行
運用・管理	フルマネージド	サーバレス	フルマネージド	サーバレス
依存関係の定義	タスクの順次実行だけでなく分岐など柔軟な定義が可能	タスク(ステップ)の順次実行だけでなく分岐など柔軟な定義が可能	変換処理(パイプライン ノード)の流れの依存関係定義がメイン。	変換処理(アクション)の順次実行定義が可能
スケジュール機能	実行間隔だけでなく開始日、終了日などの定義でより柔軟なスケジューリングが可能	Cloud Scheduler(フルマネージドのcron サービス) で定義	cron ベースでの定義 (GUI で定義可能)	cron ベースでの定義 (GUI で定義可能)
開発言語	Python でワークフローの定義	YAML でワークフローの定義	GUI でパイプラインの定義 ビルトインの変換処理も多数提供されている	主に SQL で JavaScript やビルトイン関数などを合わせてパイプラインの定義
連携可能なサービス	様々な種類のサービスと連携できるようになっている。(Operatorとして提供) カスタム Operator を作成可能	googleapi と http	多数のサービスとの連携が可能。(プラグインとして提供)カスタムプラグインの作成も可能	特定の DWH 内での処理のみ
エラー処理	タスクのリトライの柔軟な設定が可能。失敗したワークフローの再実行、途中のタスクからの実行など柔軟な再実行が可能	ステップのリトライの柔軟な設定が可能。失敗したワークフローの再実行というよりは新たに実行しなおす。	変換処理のリトライはビルトインの種類による。失敗したパイプラインは、再実行というよりは新たに実行しなおす。	リトライ処理はなし。失敗したパイプラインは再実行というよりは新たに実行しなおす。パイプラインの途中からの実行は可能
エラー通知	email 通知やエラー時の callback の定義で通知可能	単体ではエラー通知の機能はない	email、http コールでの通知	email、slack での通知
監視	様々な角度から監視できるように多数の UI を提供	実行状態の監視は明示的にログ出力が必要	実行状態をログで監視可能 変換の input output の件数も監視可能	UI のログから実行状態を監視可能

**Thank you**